ARMY RESEARCH LABORATORY

Evaluating Performance of OpenMP and MPI on the SGI Origin 2000 With Benchmarks of Realistic Problem Sizes

by Csaba K. Zoltani, Punyam Satya-narayana,
and Dixie Hisley

ARL-TR-2324 September 2000

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Abstract

Six application benchmarks, including four numerical aerodynamic simulation (NAS) codes, provided by H. Jin and J. Wu, were previously parallelized using OpenMP and message-passing interface (MPI) and run on a 128-processor Silicon Graphics Inc. (SGI) Origin 2000. Detailed profile data were collected to understand the factors causing imperfect scalability. The results show that load imbalance and cost of remote accesses are the main factors in limited speedup of the OpenMP versions, whereas communication costs are the single major factor in the performance of the MPI versions.

Acknowledgments

The authors wish to express their thanks to Haoqiang Jin and Jerry Wu for access to the beta version of the updated numerical aerodynamics simulation (NAS) benchmarks and for generously sharing their experience. Special thanks go to Gagan Agrawal and Lori Pollock of the University of Delaware for their inputs. This work was supported in part by the Department of Defense (DOD) High-Performance Computer Modernization Program.

INTENTIONALLY LEFT BLANK.

Table of Contents

INTENTIONALLY LEFT BLANK.

List of Figures

List of Tables

INTENTIONALLY LEFT BLANK.

1. Introduction

Over the last several years, several portable mechanisms for developing parallel programs have been standardized. This set includes relatively low-level libraries like the message-passing interface (MPI), parallelization directives like OpenMP, and higher level languages including high-performance Fortran (HPF). Unlike the use of vendor-specific libraries and compiler directives, these libraries and language extensions are supported on a large number of systems. At the same time, distributed shared memory (DSM) systems are emerging as an important class of parallel machines. This includes both the hardware-DSM systems like the Silicon Graphics Inc. (SGI) Origin 2000 and software-DSM systems like Treadmarks. The main advantage of such systems is that the programmers have the option of programming them using either a shared memory or message-passing paradigm or both.

In this report, an experimental study is presented to answer the following question: What are the main obstacles (among factors like communication costs, false sharing, and synchronization costs) in achieving scalable performance through each of the paradigms? Though answers have been attempted, the issue remains contentious [1]. Six benchmark programs are used, including four numerical aerodynamics simulation (NAS) codes and two irregular computational fluid dynamics (CFD) codes. The performance of OpenMP and MPI versions of these programs is examined on a 128-processor Origin 2000. Besides comparing the scalability of these versions, hardware counter-based performance data are used to understand the difference between the performance of different versions and the reasons for imperfect scalability.

In section 2, the programming environments and benchmarks used for the experimental study are explained. The results from the experiments are presented and analyzed in section 3, and the conclusions are presented in section 4.

2. Programming Environment

2.1 Origin 2000. The Origin 2000 is a DSM architecture. The machine utilized for this study is part of the U.S. Army Research Laboratory's (ARL) Major Shared Resource Center (MSRC) supercomputing assets. The largest configuration available is comprised of 128 nodes. Each processor has 1 GB of local memory. Each processor is a million instructions per second (MIPS) R12000 (R12k) 64-bit central processing unit (CPU) running at 300 MHz with two 32-kB primary caches and one 8-MB secondary cache. The older R10000 (R10k) 64-bit chips ran at 195 MHz, with two 32-kB primary caches and one 4-MB secondary cache.

An interesting aspect of the Origin 2000 system is its capability for reporting detailed profile information to the application programmers. The MIPS R12k and the older R10k are two of the very few systems in which the hardware counters are made visible to the end-users of the machine. A small set of events is monitored by the hardware counters, including cache misses, memory coherence operations, floating-point operations, and branch mispredictions. Because this monitoring is done in hardware rather than software, it is possible to extract detailed information about the state of the system without affecting the behavior of the program being monitored.

In this study, profiling data are collected by running the codes with perfex, a profiling tool that reports a count for the 32 countable event types, with no modifications to the targeted program and with only a minimal effect on its execution time. Focus was on the subset of event counts that are indicative of specific performance inhibitors to scalability. Table 1 shows the performance inhibitors that were examined and the corresponding event counts that were used to evaluate those potential problems.

2.2 Parallel Programming Environments. This study concentrated on using OpenMP as the mechanism for shared-memory programming. The Origin 2000 can also be programmed as a message-passing machine using MPI, which, like OpenMP, is portable across a number

Table 1. Using Event Counts for Performance Problem Identification

Performance Problem	Event Count
Load Imbalance	Number of floating-point operations issued per process is not comparable.
Excessive Synchronization	Number of store conditionals is high.
False Sharing	Number of store exclusives to a shared block is high.
Cache Unfriendly	L1 and L2 cache misses are high.

of platforms. MipsPro 7.2.1 compiler was used, and the applications were compiled with f77 using aggressive optimizations (-O3 flag).

2.3 Benchmarks and Problem Sizes. This study of OpenMP is focused on four of the NAS Parallel Benchmarks (NPBs), which are most relevant to the Army's applications, and two additional benchmarks, called IRREG and LES. The NPB set was developed by the NAS Program at NASA Ames Research Center for the performance evaluation of parallel computing systems [2]. NPBs mimic the computation and data movement characteristics of large-scale CFD applications. This study focused on three simulated application codes (LU solver [LU], SP, and block tridiagonal [BT]), and one kernel (conjugate gradient [CG]).

The assumption is that MPI can give the run with the least amount of computing time requirement. The NAS-optimized MPI version of the four kernels tested was then the basis for comparison. In this study, MPI implementations of the benchmarks were obtained from the NPB 2.3 NAS website [3]. The rationale behind the PBN versions given by the working team is to provide the community with an optimized version of NPB 2.3-serial and a sample OpenMP implementation. The NPB and PBN versions specify three problem sizes for the benchmarks. This report focuses on the Class B problem sizes, as they are the closest in size to realistic problem sizes, as defined by the applications commonly run at ARL. Table 2 shows the problem sizes for Class A, Class B, and Class C for each benchmark. A comparison in processing times between Class B and C is given in Figure 1.

Table 2. NPB Problem Sizes (Number of Elements)

Benchmark Code	Class A	Class B	Class C
BT	64^3	102^3	162^3
LU	64^3	102^3	162^3
Pentadiagonal Solver	64^3	102^3	162^3
CG	12,000	75,000	150,000

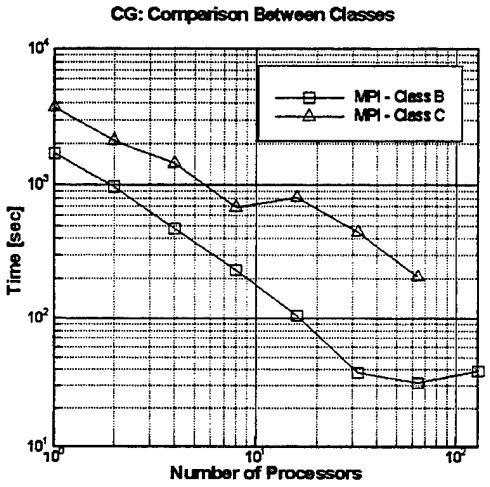

Figure 1. Timing for MPI Version of CG for Classes B and C, 12k Chip.

Another benchmark that has been focused on is the large eddy simulation (LES) [4]. LES can be used to characterize turbulent flow, where large-length scales signify the domain size and small-length scales represent dissipative eddies. Although small scales are modeled due to their isotropic nature, high-performance computing (HPC) resources are required to capture the large energy-carrying length scales. In this report, a vectorized simulation code is optimized and parallelized for Origin 2000 performance. A realistic simulation of flow past a backward-facing step with a problem size of $32 \times 32 \times 32$ is used to study scaling behavior. Periodic boundary conditions are applied in the stream-wise and span-wise directions.

The second non-NAS benchmark being examined is IRREG [5]. IRREG is abstracted from a CFD application that uses unstructured meshes to model a physical problem. The mesh is

represented by nodes, edges that connect two nodes, and faces that connect three or four nodes. For the realistic submarine mesh used in these benchmark runs, the number of nodes, edges, and faces were 92,564, 623,003, and 504,947, respectively.

3. Experimental Results

In this section, a comparison of the performance of OpenMP and MPI versions of four NAS codes and two irregular CFD codes using OpenMP is presented.

3.1 Comparing OpenMP and MPI Performance. The performance for OpenMP and MPI versions of CG, LU, SP, and BT are shown in Figures 2 and 3. The plots show wall-clock time as a function of the number of processors. In general, two observations can be made from these four plots.

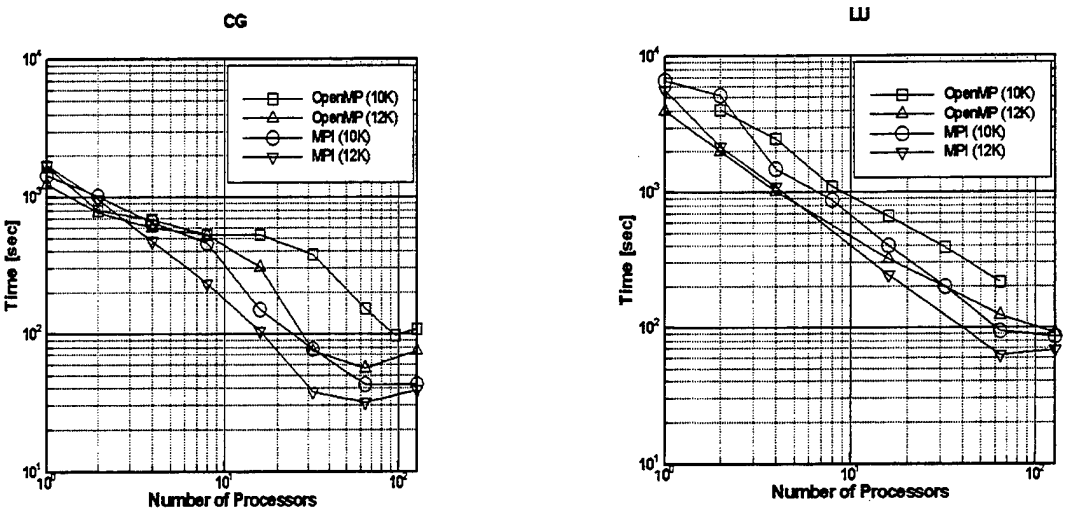

Figure 2. Scalability of OpenMP and MPI Versions of CG and LU.

Reasonably good scalability is achieved when up to 64 processors for all of the 8 programs (2 versions for each of 4 benchmarks) are used. The speedup starts leveling off for configurations beyond 64 processors, which shows that problem sizes in NAS Class B data sets are not suitable for parallelization on a very large number of processors. For three of the four applications, MPI

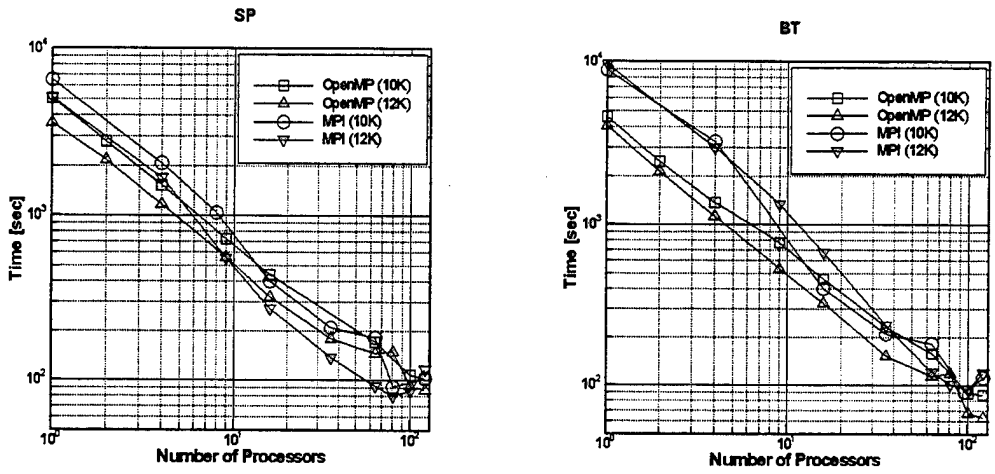

Figure 3. Scalability of OpenMP and MPI Versions of SP and BT.

achieves better performance than the OpenMP versions. The MPI versions are significantly faster for LU and CG, slightly better on large configurations for SP, and slightly slower on BT.

Of this set of benchmarks, using the R10k chip, the poorest speedups are achieved for CG. On 128 processors, the OpenMP version achieves a speedup of 14. A slightly higher speedup of 15 is achieved at the 64-processor configuration. The performance of the MPI version of CG is significantly better on 16, 32, 64, and 128 processors. On both the 64- and 128-processor configurations, the MPI version achieves a speedup of 43. For LU, OpenMP scales reasonably well until 64 processors, achieving a speedup of 32. The MPI version has significantly better speedup again, achieving a factor of 80 on 128 processors. For BT, OpenMP achieves a speedup of nearly 50 on 128 processors. The speedup of the MPI version is only 30. It should be noted that the 1-processor version of MPI performs much worse as compared to the OpenMP sequential version of this code. The results from MPI and OpenMP are the closest in the case of SP. Speedup of nearly 50 is achieved on 121 processors for both the versions.*

How profiling data from perfex can be used to determine the reasons for imperfect scalability and the differences in performance of OpenMP and MPI versions of the programs is now

* This code was executed on 121 processors because it required a square number of processors.

discussed. For a shared-memory program run on a DSM architecture, the following factors usually contribute to a lower-than-ideal speedup: load imbalance, which implies that the work in parallelized loops is not evenly distributed among the processors, and synchronization costs, which denote the time spent by the processors in coordinating the progress of the computation among themselves. False sharing occurs when two or more processors access different variables that happen to be colocated on the same cache block, with at least one of the accesses being a write. Once the write occurs, the entire cache line is invalidated to other processors. Remote accesses indicate frequent references to off-processor data, which are expensive compared to references to local data.

For the message-passing versions, the two common causes of imperfect speedup are communication costs and load imbalance. Since single-program, multiple-data (SPMD) versions of programs are run and there is no shared-memory support, false sharing and synchronization costs do not occur. For each of the eight programs in which scalability numbers have been presented, hardware counter data obtained from perfex were analyzed. For all OpenMP programs, the event counts and typical times obtained for synchronization and false sharing were extremely low (less than 3 s), even for the highest number of processors used. In general, a good level of cache friendliness was seen for all programs except CG. Cache friendliness was examined by looking at the average L1 and L2 cache hit rates returned by perfex. L2 cache hit rates were consistently higher than 0.9 for each of the eight programs, and L1 cache hit rates were also greater than 0.9 for all programs except CG. CG is an irregular code; therefore, poor L1 locality is achieved. The load imbalance issue was examined by looking at the number of floating-point operations performed over different processors in each run. A load-balanced program will have very similar numbers for the number of floating-point operations performed across all processors. Figure 4 shows the same data for OpenMP and MPI versions of CG. Detailed data from LU and SP are not presented here, but trends are explained later. Figure 5 shows the minimum, maximum, and average number of cycles spent on floating-point operations across all processors on the OpenMP and MPI versions of BT. The increase in range with an increasing number of processors suggests a problem with load balancing.

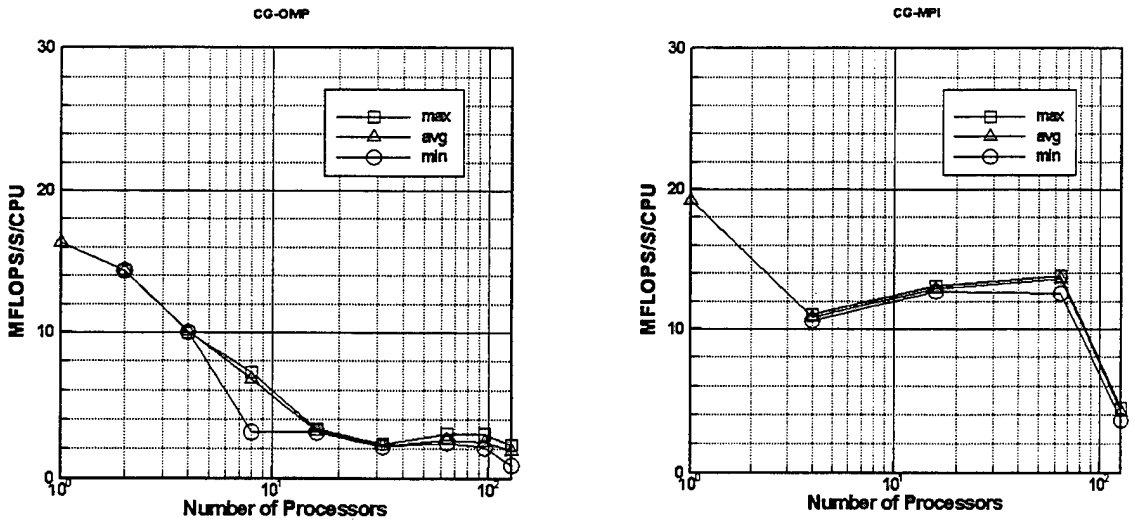

Figure 4. Max, Average, and Min MFlops Across All Processors for OpenMP and MPI for CG.

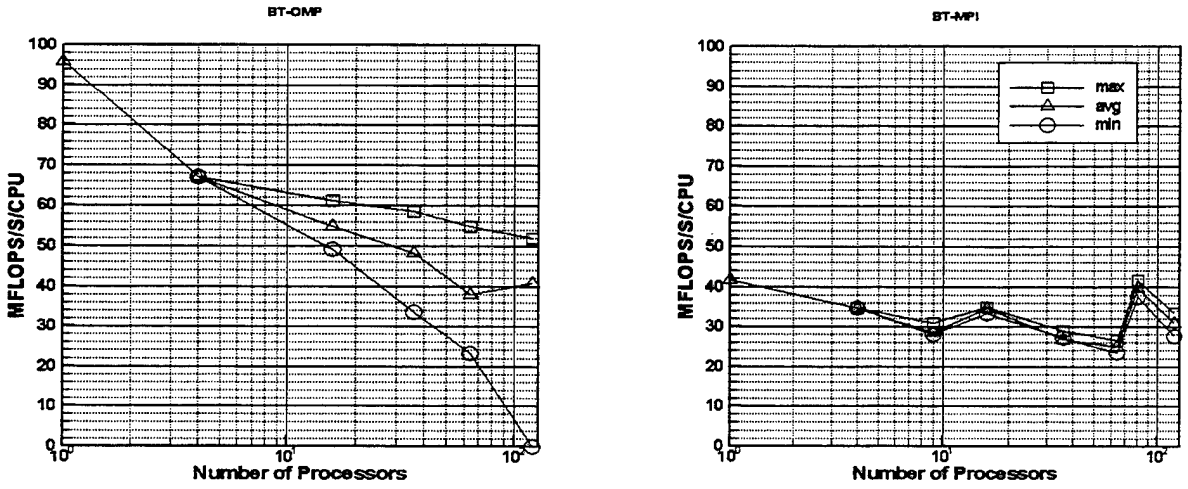

Figure 5. Max, Average, and Min MFlops Across All Processors for OpenMP and MPI for BT.

The difference in the number of floating-point operations between different processors explains the limited speedup (50 times on 128 processors) of the OpenMP version of BT. The results are very different from the MPI version of the same code. For the OpenMP version, the

load is evenly balanced between different processors on all processor configurations. Interestingly, the OpenMP version gives overall better performance than the MPI version of BT. A possible explanation for poor performance of the MPI version is the high communication costs.

The performance of the OpenMP version can be further improved by better work distribution. The program typically has nested loops where the number of iterations across each dimension is 102 (for Class B). The loop-level parallelized OpenMP version achieves parallelism across only a single dimension, so there is no way of using more than 102 processors. Possibly, by using additional directives or by using SPMD-style OpenMP parallelism, the performance of the OpenMP version of BT can be enhanced.

Similar trends are seen from CG. Excellent load balance is demonstrated by the MPI version, leading to good performance. Load imbalance can be seen for the OpenMP version, though it is not as severe as in the case of BT. Because of the irregular accesses in this code, the high cost of frequent nonlocal references is likely to be another important factor behind limited speedup. Unfortunately, perfex does not provide a mechanism for accurately measuring the number of nonlocal references. Also, remote references can be aggregated in message-passing versions, which is not possible in a shared-memory version.

In the case of SP, the OpenMP version achieves good load balance on 100 processors. The number of floating-point operations performed by each processor only ranges from 21.21×10^6/s to 18.81×10^6/s. However, on 121 processors, some of the processors do not get any work, for similar reasons as BT. Good load balance for the OpenMP version explains why the performance of the OpenMP and MPI versions is very similar.

With LU, significant load imbalance is observed with OpenMP. On 64 processors, the number of floating-point operations performed by each processor ranges from 29.63×10^6/s to 14.03×10^6/s. The load imbalance for the OpenMP version explains the difference in the performance of OpenMP and MPI versions.

3.2 Communication Cost Issues. The performance of the benchmark BT under MPI lagged under OpenMP (see Figure 3). To understand the issues involved, VAMPIR and VAMPIRTRACE (parallel processing tools from Pallas GmbH) were run. The tools give a breakdown of time spent for different tasks, including MPI, and also identify load imbalances. Runs were made with 16, 36, and 64 processors. In the latter, 65% of the total processing time was spent on MPI. The MPI runs also showed that load imbalances were present (i.e., only 9 of the processors in the 64 processor case were actually 50% occupied by the application, while in 17 of the processors, this figure was less than 25%). Improving MPI processing is feasible, but was not attempted here.

3.3 OpenMP Implementation of Irregular CFD Codes. Both of the non-NAS benchmarks, LES and IRREG, were parallelized using the SPMD style of OpenMP that relies heavily on domain decomposition. While domain decomposition can result in a coarse-grain program exhibiting good scalability, it does transfer the responsibility of decomposition from the compiler to the programmer. Once the problem domain is decomposed, the same sequential algorithm is followed but is modified to handle the multiple subdomains. The program is replicated on each thread but has different extents for the subdomains. Also, data that are local to a subdomain (not shared globally) are specified as private or thread private. Thread private is used for subdomain data that need file scope or are used in common blocks. Also, message passing is replaced by shared data that can be read by any thread.

For LES, initialization of the data is parallelized using one parallel region for better data locality among active processors. The main computational kernel is embedded in the time-advancing loop. The time loop is treated sequentially, and the kernel itself is parallelized using another parallel region. In this parallel region, the $32 \times 32 \times 32$ mesh is blocked in the z-direction and each block is tasked to a different processor.

The IRREG code contains a series of loops that iterate over nodes, edges, and faces. The loops over edges and faces involve indirect accesses to memory locations, which are difficult to analyze and parallelize in a loop-level sense. However, a parallel version of the code can be

accomplished by partitioning the nodes among the processors. The edges and faces are assigned to the processor that owns a majority of the corresponding nodes. The recursive coordinate bisection (RCB) partitioner used in the code does not optimally minimize the number of cut edges (communication effort) but does attempt to reduce the amount of communication and load balance the computational work. The performance of LES and IRREG is shown in Figure 6. A speedup of 5.1 is obtained on eight processors. In LES, the matrix solver, the most expensive module, is made cache-friendly by optimizing it for single CPU efficiency. Inherent data dependencies contribute to the imperfect scaling observed for eight processors. For IRREG, the speedup was measured on up to 32 processors. Again, the parallel version scaled quite well, with a factor of 30.0 on 32 processors. The speedup results obtained from initial attempts to parallelize IRREG using loop-level parallelization resulted in almost no speedups. Data and work distribution using specialized partitioners were extremely important for the parallel performance of this code, which could not be achieved through directives for loop-level parallelism.

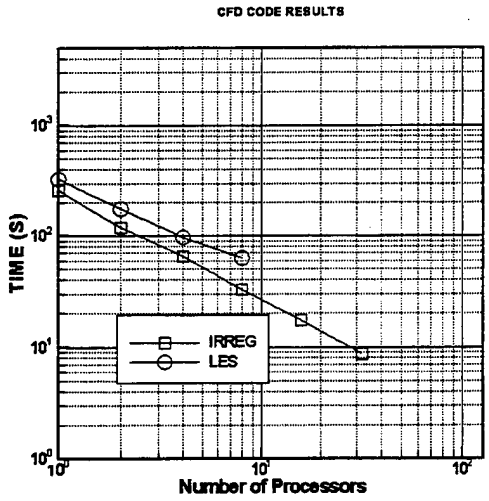

Figure 6. Performance of OpenMP of LES and IRREG With the R10k Chip.

4. Conclusions

In this report, experiments have been conducted to study the performance achieved through shared-memory (OpenMP implementations) and message-passing (MPI implementations) paradigms for six benchmark programs with realistic problem sizes run on a 128-processor Origin 2000 with both the older R10k and the newer R12k chips. Moreover, hardware-profiling data were analyzed to understand the reasons for imperfect speedups of these codes.

These experiments lead to several interesting observations. A somewhat better performance was obtained from MPI programs, as compared to the OpenMP. The main factor behind limited scalability of the OpenMP versions was load imbalance. Only the outer loops were parallelized, and, on large configurations, not all processors could be kept busy. The second most important performance obstacle for OpenMP versions was the cost of remote references. False sharing and synchronization costs were insignificant for the programs in this benchmark set.

The MPI versions demonstrated excellent load balance, with parallelism obtained through domain decomposition. The main factor in the limited scalability of MPI versions was communication costs. The MPI codes' communication costs are indeed higher as shown by VAMPIR TRACE data. This experience in developing parallel versions of two irregular CFD codes found that the SPMD style parallelization facility of OpenMP enabled easy and efficient parallelization of these applications.

This study concluded that programmers need to concentrate on achieving good work distribution while optimizing the performance of OpenMP versions, and they need to concentrate on improving communication performance while optimizing the performance of MPI versions. These conclusions are applicable only to the programs that have similar features to the benchmark programs studied here.

5. References

1. Nikolopoulos, D. S., and T. S. Papatheodorou. "A Comparison of MPI, SHMEM and Cache-Coherent Shared Address Space Programming Models on the SGI Origin 2000." International Conference on Supercomputing, June 1999.

2. Bailey, D., T. Harris, W. Saphir, R. vander Wijngaart, A. Woo, and M. Yarrow. "NAS Parallel Benchmarks 2.0." Technical Report NAS-95-020, NASA Ames Research Center.

3. Numerical Aerodynamic Simulation Program. www.nas.nasa.gov/cgi-bin/softwave/start. NASA Ames Research Center, 1999.

4. Wang, W. P. "Coupled Compressible and Incompressible Finite Volume Formulations of the Large Eddy Simulation of Turbulent Flows With and Without Heat Transfer." PhD thesis, Iowa State University, 1995.

5. Das, R., D. J. Mavriplis, J. Saltz, S. Gupta, and R. Ponnusamy. "The Design and Implementation of a Parallel Unstructured Euler Solver Using Software Primitives." *AIAA Journal*, vol. 32, no. 3, pp. 489–496, March 1994.

INTENTIONALLY LEFT BLANK.

NO. OF COPIES	ORGANIZATION
2	DEFENSE TECHNICAL INFORMATION CENTER DTIC DDA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	HQDA DAMO FDT 400 ARMY PENTAGON WASHINGTON DC 20310-0460
1	OSD OUSD(A&T)/ODDDR&E(R) R J TREW THE PENTAGON WASHINGTON DC 20301-7100
1	DPTY CG FOR RDA US ARMY MATERIEL CMD AMCRDA 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN PO BOX 202797 AUSTIN TX 78720-2797
1	DARPA B KASPAR 3701 N FAIRFAX DR ARLINGTON VA 22203-1714
1	NAVAL SURFACE WARFARE CTR CODE B07 J PENNELLA 17320 DAHLGREN RD BLDG 1470 RM 1101 DAHLGREN VA 22448-5100
1	US MILITARY ACADEMY MATH SCI CTR OF EXCELLENCE MADN MATH MAJ HUBER THAYER HALL WEST POINT NY 10996-1786

NO. OF COPIES	ORGANIZATION
1	DIRECTOR US ARMY RESEARCH LAB AMSRL D D R SMITH 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRL DD 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CI AI R (RECORDS MGMT) 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI LL 2800 POWDER MILL RD ADELPHI MD 20783-1145
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CI AP 2800 POWDER MILL RD ADELPHI MD 20783-1197
	ABERDEEN PROVING GROUND
4	DIR USARL AMSRL CI LP (BLDG 305)

NO. OF COPIES	ORGANIZATION
18	DIRECTOR US ARMY RESEARCH LAB AMSRL CI N RADHAKRISHNAN AMSRL CI CT A CELMIŅŠ AMSRL CS TT J POLK AMSRL CI H W STUREK C NIETUBICZ P SATYA NARAYANA AMSRL CI S A MARK AMSRL CI HA C ZOLTANI (4 CPS) D HISLEY D PRESSEL R NAMBURU AMSRL WM B A HORST E SCHMDT AMSRL WM BC P PLOSTINS H EDGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2000	Final, Oct 98–Sep 99

4. TITLE AND SUBTITLE
Evaluating Performance of OpenMP and MPI on the SGI Origin 2000 With Benchmarks of Realistic Problem Sizes

5. FUNDING NUMBERS
423612.000

6. AUTHOR(S)
Casba K. Zoltani, Punyam Satya-narayana, and Dixie Hisley

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-CI-HC
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-2324

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Six application benchmarks, including four numerical aerodynamic simulation (NAS) codes, provided by H. Jin and J. Wu, were previously parallelized using OpenMP and message-passing interface (MPI) and run on a 128-processor Silicon Graphics Inc. (SGI) Origin 2000. Detailed profile data were collected to understand the factors causing imperfect scalability. The results show that load imbalance and cost of remote accesses are the main factors in limited speedup of the OpenMP versions, whereas communication costs are the single major factor in the performance of the MPI versions.

14. SUBJECT TERMS
benchmarking, parallel processing, OpenMP

15. NUMBER OF PAGES
22

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number/Author ARL-TR-2324 (Zoltani) Date of Report September 2000

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name E-mail Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)